European Sixth Framework Network of Excellence FP6-2004-IST-026854-NoE

Deliverable D7.5

Large Scale Management Interim Report

The EMANICS Consortium

Caisse des Dépôts et Consignations, CDC, France
Institut National de Recherche en Informatique et Automatique, INRIA, France
University of Twente, UT, The Netherlands
Imperial College, IC, UK
Jacobs University Bremen, JUB, Germany
KTH Royal Institute of Technology, KTH, Sweden
Oslo University College, HIO, Norway
Universitat Politecnica de Catalunya, UPC, Spain
University of Federal Armed Forces Munich, CETIM, Germany
Poznan Supercomputing and Networking Center, PSNC, Poland
University of Zürich, UniZH, Switzerland
Ludwig-Maximilian University Munich, LMU, Germany
University of Surrey, UniS, UK
University of Pitesti, UniP, Romania

For more information on this document or the EMANICS Project, please contact:

Dr. Olivier Festor
Technopole de Nancy-Brabois - Campus scientifique
615, rue de Jardin Botanique - B.P. 101
F-54600 Villers Les Nancy Cedex
France
Phone: +33 383 59 30 66
Fax: +33 383 41 30 79
E-mail: <olivier.festor@loria.fr>

Document Control

Title:	Large Scale Management Interim Report
Type:	Public
Editor(s):	Ramin Sadre
E-mail:	sadrer@cs.utwente.nl
Author(s):	WP7 Partners
Doc ID:	D7.5

AMENDMENT HISTORY

Version	Date	Author	Description/Comments
0.1	2007-07-03	H. Tran, J. Schönwälder	Initial version of a LaTeX template
0.2	2009-06-16	R. Sadre	Initial version for integration
0.3	2009-06-22	R. Sadre	FMAD activity description
0.4	2009-06-28	R. Sadre	Other activity descriptions added

Contents

1 Executive Summary

Scalability is one of the major issues in network management. Many management solutions that are available for small networks cannot be easily applied to networks consisting of hundreds or thousands of communication entities.

The work in this work-package has been driven by five activities that started in January 2009 as a result of an open call for proposals, structured around two themes: security in large-scale environments and monitoring and configuration in large-scale environments. This interim deliverable reports the scientific results achieved in the period of January 2009 – June 2009 in work-package 7 of the Emanics NoE. It will be completed by the final report D7.6 at the end of 2009.

2 Introduction

Scalability is one of the major issues in network management. Many management solutions that are available for small networks can not be easily applied to networks consisting of hundreds or thousands of communication entities. The research activities in this work package focus on the security, configuration and monitoring in large-scale environments. Work is structured around the following two tasks, that emerged from an open call for proposals:

- **T7.1 Security in large-scale environments:** Activities in this task address the design, implementation and evaluation of new techniques to manage the security in large-scale environments, comprising the detection of intrusions and attack attempts, the secure access to resources, etc.
- **T7.2 Monitoring and configuration in large-scale environments:** Large-scale networks pose several challenges for management, for example the large amount of traffic transported in such networks, the large and variable number of participating network entities and involved parties, etc. Activities in this task follow different approaches such as distribution and virtualization to allow the effective monitoring and configuration under such conditions.

An open call for activity proposals was initiated at the end of 2008 and finalized at the beginning of 2009. Five activities have been chosen to be funded during the last phase of the EMANICS project.

In this document, the goals of those activities are described and the progress achieved in the period of January 2009 – June 2009 is reported. For an overall view on the activities and the involved participants, we summarize them in the following:

- **BioScale II:** Extended Scalable Management of Biometric devices (UniZH, UniBwM)
- **FMAD:** Flow-based Monitoring and Anomaly Detection (UT, PSNC, JUB, UniZH, INRIA, UniBwM, UCL)
- **MaMANET:** Management of large MANETs (UniBwM, UZH, JUB)
- **SeMI:** Security Management Infrastructure (UniBw, Jacobs)
- **VirtMon:** Virtualization Monitoring (UPC, UCL, UniZH)

Note that this document is an interim report that will be completed by the final report D7.6 at the end of 2009.

3 Extended Scalable Management of Biometric devices (BioScale II)

3.1 Introduction

Biometric devices emerged to a mature technology and begin to fulfill its promises for physical access control and login to computers. But with today's device specific software provided by manufacturers, medium and large scale deployments of biometric devices is unfeasible. One of the main challenges of such large scale deployment projects is a scalable distribution system for biometric templates. The objective of the BioScale I and II activity is the development of a scalable management approach for biometric devices.

3.2 Activity Description

In this section the goals of the BioScale I and II approach will be described sketchy to give a general survey of the BioScale project as a whole:

3.2.1 BioScale I

As mentioned in the final report of BioScale I a scalable template distribution strategy for biometric devices in a large scale deployment has been defined and implemented. For this the template distributer of the application BioXes was reused and to make sure that the scalability tests were feasible the template distributer had to be changed into a prioritized template distributer. On the other hand a dummy device had to be implemented to make it possible to test the above mentioned distribution strategy. During the implementation of the initial BioScale I approach a set of additional requirements emerged, which have to be considered for a proper final design of a scalable distribution system in a large scale deployment of biometric devices. Therefore, to achieve a full-fledged and complete approach for a scalable management approach for biometric devices in a large scale environment, the BioScale II approach emerged.

3.2.2 BioScale II

As just mentioned additional requirements emerged in the BioScale I approach. Because of these newly detected requirements a small subset of existing tasks of the initial BioScale I approach had to be postponed and adopted to the BioScale II approach. For the BioScale II approach the following task have to be finished:

- Task 1: Set up test scenario and scalability tests.
- Task 2: Analysis of measurement results.
- Task 3: Comprehensible User Interface.
- Task 4: Distributed User Interface.

3.3 Strategy for BioScale II

In this section the tasks of the BioScale II approach are described in more detail and the strategy to fulfill the tasks is shown.

3.3.1 Task 1: Test scenario and scalability tests

Based on the BioScale I implementation's dummy device and its multiple instances the test scenario has to be defined and set up as well as tests have to be performed to show that the approach designed for the template distribution is scalable in terms of:

- time required for distribution
- bandwidth usage during distribution
- respond time of devices during template distribution
- integrity of distribution

3.3.2 Task 2: Analysis of measurement results

Those results obtained from the tests performed in Task 1 have to be analyzed. Based on these results of the analysis a policy for the template distribution in a large-scale environment has to be defined and fully implemented in turn.

3.3.3 Task 3: Comprehensible User Interface and Visualization

Especially in large-scale installations a comprehensive user interface is important in order to enable the user keeping the overlook. Designing such a user interface, which allows the user to manage a huge number of biometric devices and configure access rights for the distribution plan, respectively, is a challenging task. Therefore, the objective of task 2 is to design such a user interface and implement it prototypically for visualizing large-scale installations. In an iterative process the design of the interface will be improved by real-life tests.

3.3.4 Task 4: Distributed User Interface

Large-scale installations of biometric devices are mostly distributed, i.e., consisting of installations at different sites. The possibility of a distributed management is in this case of utmost importance. Centrally managing such installations would limit the scalability, not from a technical perspective, but from an organizational one. The key objective of this task is defined as designing relevant mechanisms for allowing several operators to administer biometric identities at the same time. The key technology to apply is to investigate Web 2.0 mechanisms and use those for biometric device and network management, where applicable. In case of functional gaps identified, different, proprietary solutions will be developed.

3.4 Course of Action and Progress Report for BioScale I and BioScale II

3.4.1 Task 1: Set up the test scenario and perform the scalability test

In order to have several thousand users to perform a real life test an LDAP integration has been implemented and tested. Next step is to set up the scenario with several thousand devices and users.

3.4.2 Task 2: Analyze measurement results

Since this task depends on the results of task 1 no action has taken place here.

3.4.3 Task 3: Comprehensible user interface and visualization

A second prototype with the adapted feedback has been implemented and a second round of field tests has been carried out. Based on the feedback the final screen for a comprehensible user interface has been designed and will be implemented in a next step.

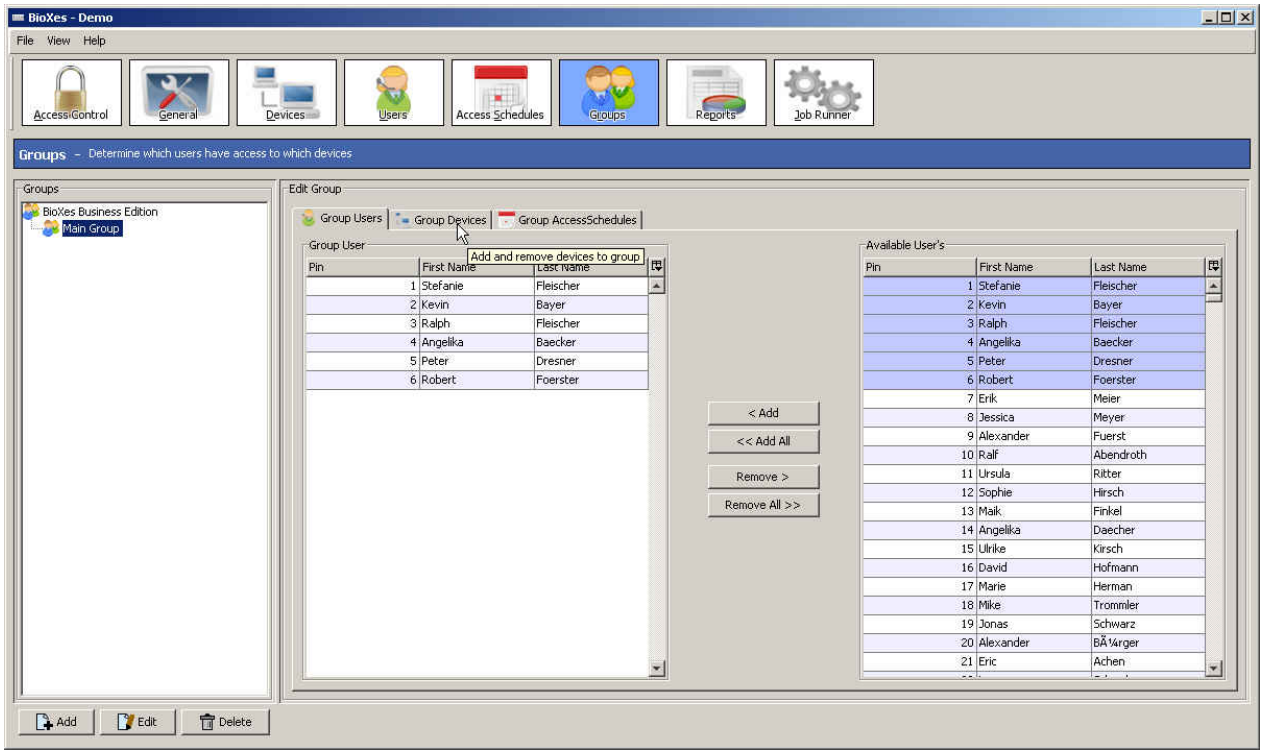


Figure 1: In this picture the old version of the user interface of the distribution plan is shown.

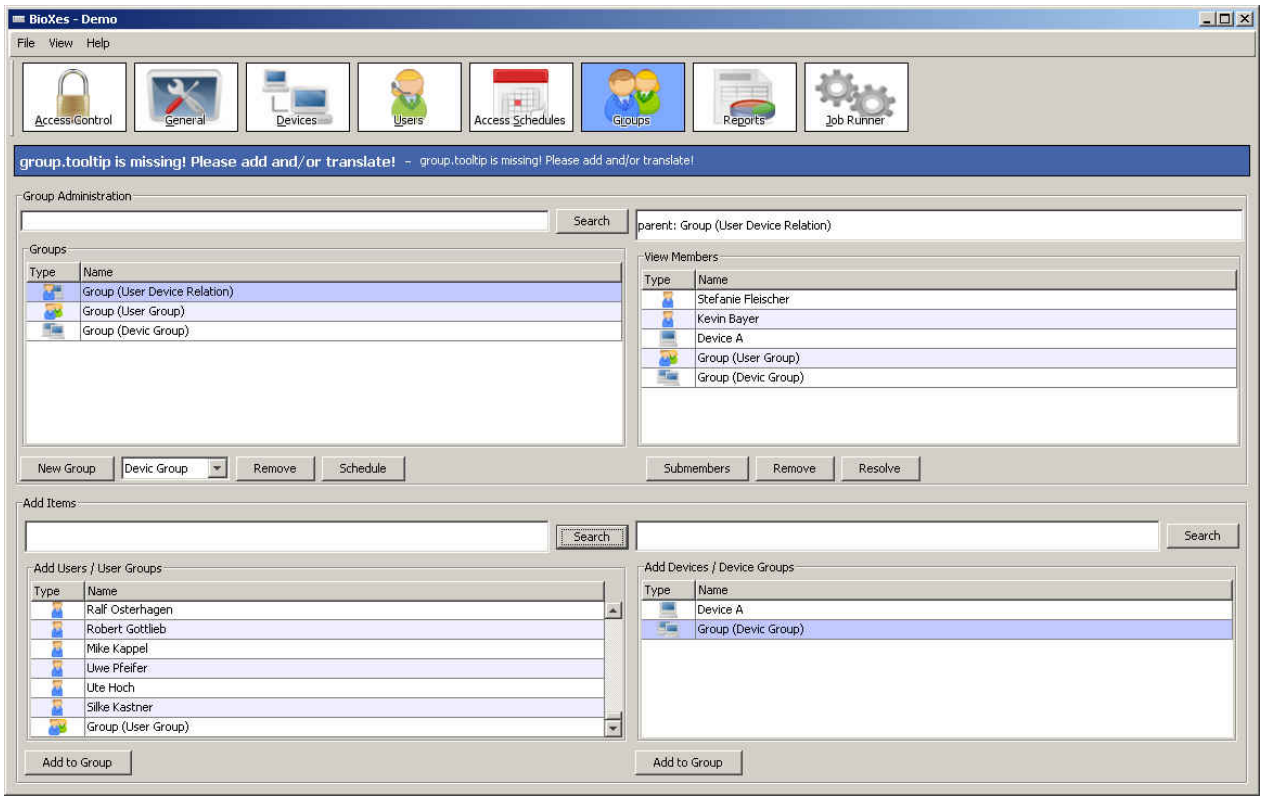


Figure 2: In this picture the second version of the user interface of the distribution plan is shown.

3.4.4 Task 4: Distributed User Interface

The distributed user interface had to be redesigned and a first prototype of the application architecture has been set up. This is because an adequate framework has been found: GWT-Mosaic. The implementation of the other screens has started.

4 Flow-based Monitoring and Anomaly Detection (FMAD)

4.1 Introduction and activity description

The spread of Gbps networks and the constantly increasing amount of traffic transported by such networks has drawn the attention to scalability aspects of the management of such networks, leading to an interest in flow-based techniques for analysis and anomaly detection in large and/or fast networks. This activity is a continuation of the work done in WP7 on flow-based monitoring and analysis techniques. In particular, the following topics are of interest:

- Develop and evaluate flow-based techniques for the detection of anomalies, especially attacks.
- Searching flow pattern using a novel stream-based flow-record query language.
- Coordination of flow trace collection in a large scale network.
- Joint packet/code/flow level malware and attacks collection and analysis.
- Fingerprinting of network environments for a fast and reliable detection of, possibly malicious, changes and manipulations in the network structure taking into account a minimization of the false-alarm-rate.
- Traffic characterization of a NREN with regard to application/protocols level in research traffic.
- In sampled monitoring data, it is difficult to recover estimates of the distribution of lengths of TCP flows. We propose to address this problem using a technique known as sample-and-hold. It is possible that anomalies in the flow length distribution (for example a large number of very short flows) could be used to detect unusual traffic (for example a SYN flood attack).

In Section 4.2 of this report, it is investigated if spammers can be detected at the network level, based on just flow data. This problem is challenging, since no information about the content of the email message is available. A spam detection algorithm is proposed, which is able to discriminate between benign and malicious hosts with high accuracy. Please note that the achieved results have been also published in [1].

In Section 4.3, this report introduces to traffic characterization, a sub-activity which was initiated during this reporting period. Final results will come in the next deliverable.

4.2 Detecting spam at the network level

Spam is a problem that all Internet users experience in their everyday lives. Symantec Corporation estimates that over 80% of all emails sent in 2008 were spam, a trend that, with a touch of irony, the company considers to be “normal” [2]. The reason we are constantly

flooded with unsolicited messages is that spam is profitable. As such, spam detection is likely to remain an "open battlefield" in the coming years.

Nowadays, the most common countermeasures against spam are spam filters. Mail servers usually host the core of spam filtering operations: tools such as Spamassassin [3] reject or accept email messages based on their content. Moreover, many mail clients also locally scan the user's inbox. However, spam messages are designed to look similar to legitimate emails: examples are "phishing" emails that ask you to provide your bank details. Such camouflaging behavior reduces the effectiveness of content-based methods.

We propose a spam detection approach that does not rely on content information. More specifically, our contribution is based on network flows, defined as "a set of IP packets passing an observation point in the network during a certain time interval and having a set of common properties" [4]. These common properties typically include source/destination addresses/ports and protocol type, and they unequivocally define a flow. Flows have recently received great attention in the research community [5], since they allow scalable network monitoring of large infrastructures. Flows typically only report information about the amount of packets and bytes exchanged during a connection, but nothing about the content of the communication.

In this context, spam detection is a challenge. This section aims to address the following question: *Is it possible to detect hosts from which spam originates by using just flow data?* More specifically, we want to investigate (a) if spam differs from legitimate SMTP traffic at the flow level and (b) how to detect spam at the flow level.

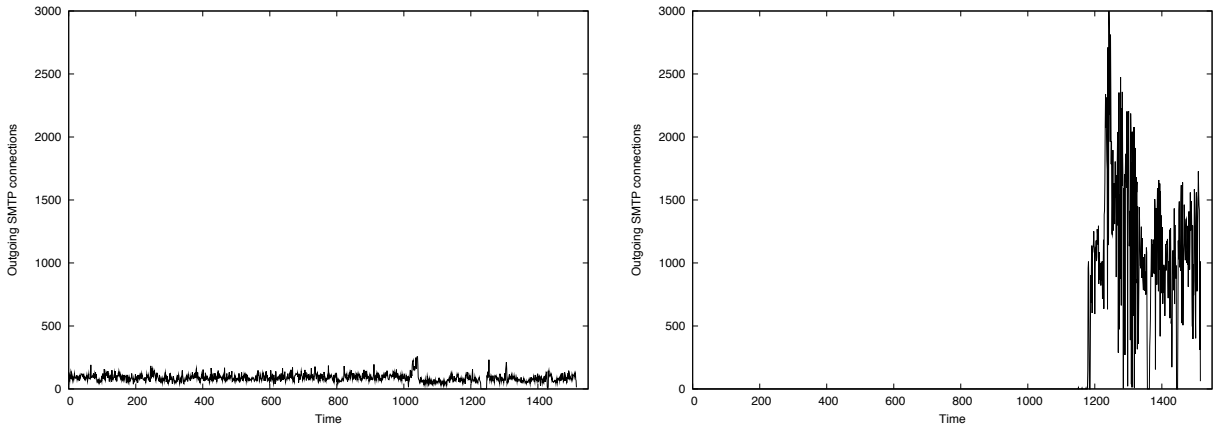
The general assumption in the research community is that a spammer host will behave differently from a legitimate mail server [6, 7, 8]. Capturing this behavior at the network level can lead to the development of powerful tools for early spam detection, easing both the server-side load and the filtering in the client. One contribution in this field is the work of Desikan et al. [9], in which the analysis of time-evolving SMTP connection graphs helps distinguish between mail servers and spammers. A different approach is that taken by Ramachandran et al. [6]. The authors' assumption is that the network behavioral patterns of a spamming host are far less variable than the spam content itself. They therefore propose a spam detection approach based on automatic clustering and classification of sender IP addresses that show a similar behavior over a short observation time. More attention to flow approaches has been given in the works of Schatzmann et al. [7, 8] and Cheng et al. [10]. In [7, 8], the authors suggest that the average number of bytes, packets and bytes/packets of failed, rejected and accepted connections are flow properties suitable for the classification of spam flows. The authors rely on server logs for flow classification. On the other hand, in [10], the authors propose an alternative definition of flows that allows the stateful analysis of spam traffic. Finally, Žádník et al. [11] propose the use of classification trees for spam identification based on flow characteristic.

Compared to the previously mentioned contributions, we propose a spam detection algorithm that relies on Netflow compatible flow data and allows the detection of spamming hosts based on just network characteristics.

This section is organized as follows. Section 4.2.1 describes SMTP traffic from a flow perspective, highlighting the differences between a normal and a suspicious host. In Section 4.2.2 we present our spam detection algorithm, followed by a validation of our approach in Section 4.2.3. Conclusions are drawn in Section 4.2.4.

4.2.1 SMTP traffic at the flow level

It is a common assumption that a spamming host's behavior will differ from legitimate SMTP servers. Yet it is interesting to see if this assumption holds in real traffic.



(a) (b)

Figure 3: Flow level behavior for a university mail server (a) and a suspicious machine (b)

The University of Twente, for example, relies on a system of five load-balanced mail servers, all of them having a similar behavior. Figure 3(a) shows the outgoing SMTP traffic time-series of one of them. Each time slot on the x-axis corresponds to a 5 minutes interval, for a total of five days of observation. There is one main aspect in the mail server behavior. The mail server presents a rather constant activity baseline at around 100 connections per time slot that rarely rises above 250 connections per time slot. This aspect is very significant in our case since it shows that a legitimate mail server is characterized by a steady level of usage. Figure 3(b), on the other hand, shows the outgoing SMTP traffic time series for a host known to have sent spam. Its network behavior is totally different from the one in Figure 3(a): the time series is characterized by sudden and prolonged activity peaks and a long period in which there is no traffic. Moreover, no usage baseline is present, at the contrary of the mail server. A deeper analysis of the spammer host behavior also reveals that there is no incoming traffic, suggesting that the host has no real traffic exchanges. This behavior is commonly observed in other hosts that have sent spam.

This example suggests that the behavior of suspicious hosts differs substantially from that of legitimate mail servers. Parameters such the incoming and outgoing traffic, as well as the widely variable level of usage can be useful in defining an algorithm for automatic spam detection.

4.2.2 An algorithm for spam detection

In the previous section, we showed qualitatively that the network behavior of suspicious and legitimate hosts could be very different. We now propose an algorithm that will detect, based on just flow information, hosts that are most likely to be spammers. The algorithm consists of two main phases and a post-processing step. In the first phase, hosts that do not satisfy three basic selection criteria are filtered out. This phase aims to reduce the

amount of data to be analyzed and to improve the overall performance of the algorithm. The hosts selected in the first phase are then ranked in the second phase by means of five ordering criteria according to their likelihood of being spammers. Finally, ranked hosts are once again filtered according to a post-processing criterion. The algorithm analyzes the SMTP traffic sent and received from the network that is monitored. Of course, this means spam traffic generated by a spammer outside the monitored network and targeting a different network cannot be considered for the analysis. However, the results show that it is not necessary to have a complete overview of all the traffic generated by a spammer to achieve a good detection level.

Selection criteria The selection criteria allow us to concentrate, in the second phase, on a smaller subset of hosts. Therefore, in order to be further analyzed, a host has to satisfy all the selection criteria. The selection criteria aim to filter out at an early stage the majority the not-malicious clients. These criteria are defined as follows:

SC$_1$ Number of outgoing connections: We only select hosts that exhibit a certain level of activity:

$$\text{number of outgoing SMTP connections} > \theta_1 \tag{1}$$

SC$_2$ Connection ratio: A host is suspicious if it sends far more than it receives. The connection ratio criterion is defined as:

$$\frac{\text{number incoming SMTP connections}}{\text{number of outgoing SMTP connections}} < \theta_2 \tag{2}$$

SC$_3$ Number of distinct destinations: Criterion **SC$_1$** could also flag as suspicious a host that relies on SMTP as logging mechanism (as a printer, for instance). Such a host would probably not receive any message. Nevertheless, such host would usually report to only a limited number of destinations, while a spammer would typically diversify its destinations. A threshold for the minimum number of distinct destinations is used for discriminating these cases:

$$\text{number of distinct destinations} > \theta_3 \tag{3}$$

Ordering criteria Once the suspicious hosts have been selected by applying the selection criteria, we apply the ordering criteria to rank them according to their likelihood of being spammers. While the selection criteria are combined into a binary decision, the ordering criteria yield values from a to e that are later combined into a total score for each host.

OC$_1$ Number of incoming connections: This criterion is a refinement of **SC$_2$**. We assume that spammers are not interested in receiving SMTP connections. Therefore, a host that does not have any incoming connection is more likely to be a spammer than one that has incoming SMTP traffic. The score is calculated as follows:

$$a = \begin{cases} 1 & \text{if number of incoming SMTP connections} = 0 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

OC_2 Number of distinct destination: This criterion is a refinement of **SC_3**. We assume that a spammer would try to diversify its destinations. Therefore, hosts with a high number of distinct destinations are suspicious. We define the score b as:

$$b = \begin{cases} 1 & \text{if number of distinct destination servers} > \theta_4 \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

OC_3 Percentage of idle time: We assume that hosts with long idle periods are more suspicious than hosts that communicate more regularly over time. We define the score c as:

$$c = \text{percentage of idle time} \quad (6)$$

OC_4 Irregularity in activity: Our studies suggest that a suspicious host tends to have a highly irregular transmission pattern. We assume that a host that has a high standard deviation σ of the number of outgoing SMTP flows per 5 minute time slot is more suspicious than one with a low one. We define the score d as:

$$d = \begin{cases} 1 & \text{if } \sigma > \theta_5 \\ 0 & \text{otherwise} \end{cases} \quad (7)$$

OC_5 Number of peaks: We assume a suspicious host to show sudden traffic peaks. We define peaks as time slots where the number of outgoing connections is higher than $(\mu + k \cdot \sigma)$. μ and σ are respectively the mean and the standard deviation of the number of outgoing connections per 5 minute time slot for the host, and k is a parameter that influences the sensitivity of the measure. Hosts with a high number of peaks are therefore more suspicious. We define the score e as:

$$e = \begin{cases} 1 & \text{if } |\{\text{slots where connection rate} > (\mu + k \cdot \sigma)\}| > \theta_6 \\ 0 & \text{otherwise} \end{cases} \quad (8)$$

The detection algorithm Algorithm 1 presents the pseudocode for the detection procedure. As explained earlier, the first phase filters hosts according to the three selection criteria (lines 4 through 6). However, in order to keep the algorithm efficient, we only consider the n most active hosts, in terms of outgoing connections, that satisfy the criteria (lines 3 and 7).

In the second phase, the hosts are scored and ordered according to the ordering criteria (lines 11 through 17). For the overall score v, we calculate the average of the single scores a through e. While ranking the hosts, the algorithm also selects a subset of them that, in conjunction with the ranking, are most likely to be spammers. More specifically, only hosts that are not involved in any traffic exchange for the majority of the observation time γ are considered (line 13). This filtering permits the discrimination between hosts that have a fairly constant behavior and hosts that only transmit in bursts, as for example the hosts in Figure 3.

Finally, the algorithm only reports the m top ranked hosts (line 18). The parameter m allows tuning of the output according to the desired security level.

Algorithm 1 Spam detection procedure

procedure SpamDetection(Q : host set)
$S_1 = \emptyset$; $S_2 = \emptyset$;
for all $x \in Q$ ordered by decreasing number of outgoing connections **do**
  if x satisfies $\mathbf{SC_1} \wedge \mathbf{SC_2} \wedge \mathbf{SC_3}$ **then**
    $S_1 := S_1 \cup \{x\}$;
  end if
  if $|S_1| = n$ **then**
    break;
  end if
end for
for all $y \in S_1$ **do**
  Compute $v := \frac{1}{5} \cdot (a + b + c + d + e)$;
  if $c > \gamma$ **then**
    $S_2 := S_2 \cup \{y\}$;
  end if
end for
Order elements in S_2 by decreasing value of v;
return top m elements in S_2;

4.2.3 Experimental results and validation

Validation approach Since we based our algorithm on flows, no information about the content of the SMTP connections is available. We therefore need to rely on external services in order to evaluate our results. DNS blacklists (DNSBL) are Internet services that publish lists of offending IP addresses: in our context, IPs that have been involved in spamming activities. Spam DNSBL are repositories which content is likely to rapidly change over time: indeed, a blacklisted host can be rehabilitated if it is no longer involved in spamming activities for a sufficiently long period. Iverson [12] periodically monitors the most commonly used DNSBL and reports on their reliability.

We selected five DNSBL as trusted sources for validation: `zen.spamhaus.org`, `bl.spamcop.net`, `safe.dnsbl.sorbs.net`, `psbl.surriel.com`, and `dnsbl.njabl.org`. We chose this set of DNSBL because they clearly indicate under which conditions a host is going to be added and removed from the list. We define a host to be *positively validated* if it has been blacklisted in at least one of the five DNSBL we are considering.

Experimental settings and results We evaluate our algorithm over three data sets collected at the University of Twente: a reference data set used to developed the algorithm and two newly collected data sets referred as *Set 1* and *Set 2* in the following. Each data set spans over a period of seven days, with an average of $\sim$15M flows. The time windows over which the data sets span are not overlapping. The implementation of our approach uses SQL scripts and can process a data set in a period of 5 hours.

In our experiments, we measure the *accuracy* of the method, defined as:

$$accuracy = \frac{|\{\text{positively validated}\}|}{m} \tag{9}$$

where m is the number of hosts reported as output by the algorithm and it can be set according to the desired security level. We decided not to compute the false positive and false negative rate since it is not possible to establish a ground truth. For the hosts that we report as suspicious and that are not listed in any DNSBL, indeed, we are unable to say if they are (a) spammers that are not yet listed (true positive) or (b) normal hosts (false positive).

Parameter	Value	Parameter	Value	Parameter	Value	Parameter	Value
θ_1	200	θ_2	0.005	θ_3	5	θ_4	10
θ_5	1	θ_6	50	k	5	γ	80%

Table 1: Criteria parameter settings chosen for the experiments

Table 1 shows which parameter values have been used in the experiments. The parameters have been manually tuned based on the statistical properties of the reference data set. We measured that only 5% of the hosts we analyzed have more than 200 connections ($\theta_1 = 200$). In a similar way, only 1% of the hosts have more than 10 distinct destinations ($\theta_4 = 10$). Less than 20% of the hosts present a standard deviation $\sigma > 1$ of the number of outgoing connections per time slot ($\theta_5 = 1$). Moreover, only 1% of the hosts have more that 50 peaks (for $k = 5$, $\theta_6 = 50$). The remaining parameters are specific to the network we are analyzing. In particular, as said in Section 4.2.1, the University of Twente relies on 5 mail servers. Therefore, we set $\theta_3 = 5$. In our network, high volume SMTP sources might receive a limited amount of incoming connections ($\theta_2 = 0.005$) and, for an observation window of seven days, spammers have shown to be idle for at least 80% of the time ($\gamma = 80\%$). Finally, the algorithm selects n = 20,000 hosts that satisfy the selection criteria and outputs the top m = 100 hosts according to the ordering criteria.

Our experimental results show that, on average, the accuracy of the system is 92%. Table 2 presents the detection accuracy for each of the considered data sets. We observe that our algorithm reaches an overall accuracy of 99% in the reference data set, while the accuracy slowly decreases in the newly collected data sets. This phenomenon suggests that the parameters chosen for our experiments might need to be periodically re-tuned according to spam flow characteristics.

Criteria impact In Section 4.2.2 we introduced the criteria we used in our detection algorithm. We now evaluate the impact of each single criterion to the overall detection accuracy of the algorithm. We start evaluating the impact of the only selection criteria $\mathbf{SC}_1$ and incrementally add one criterion at each run. We measure the overall accuracy on the data set *Set 1* and *Set 2* presented in Table 2. Figure 4 shows the average trend of the accuracy curve with respect to the number of applied criteria. The error bars indicate the standard deviation of the number of validated hosts w.r.t. *Set 1* and *Set 2*.

Data set	Time window	Accuracy
Reference set	18 – 24 November 2008	99%
Set 1	2–7 April 2009	96%
Set 2	8–14 April 2009	81%

Table 2: Detection accuracy for the considered data sets.

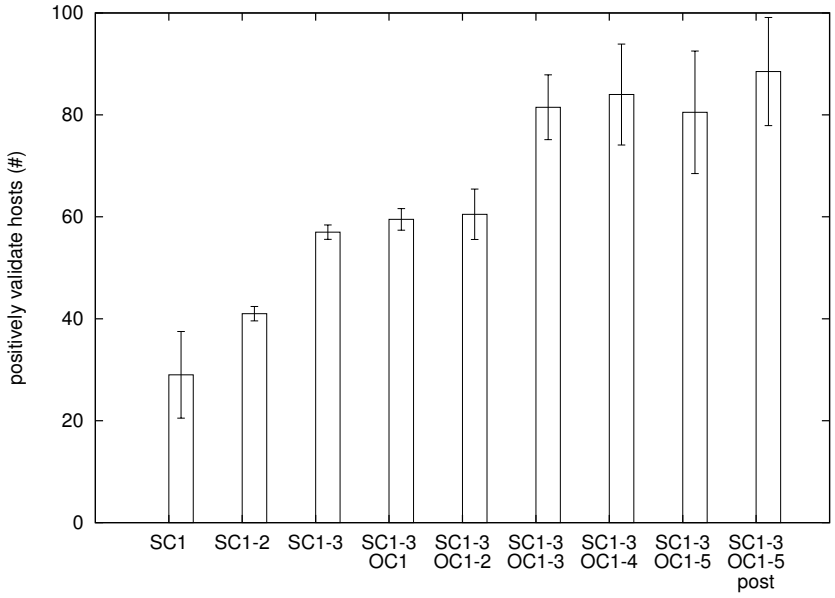


Figure 4: Impact of the selection and ordering criteria on the overall accuracy

Selection criteria $\mathbf{SC}_1$ and ordering criteria $\mathbf{OC}_3$ have the most impact on the accuracy measure, meaning that a high number of connections in a short period of time (bursts) is a key characteristic of a spamming host. Moreover, the accuracy measure presents an increasing trend, meaning that each criterion is beneficial to the detection process. The only exception is $\mathbf{OC}_5$: in data set *Set 1*, indeed, the criterion forces a decrease of the accuracy, suggesting that under certain condition this criterion may report false positives (legitimate hosts flagged as spammers).

4.2.4 Conclusions

This section has investigated if it is possible to detect spammers at the flow level, without relying on email content. Our findings show that the network behavior of suspicious hosts differs substantially from that of a legitimate mail server, both in activity level and incoming/outgoing traffic patterns. Based on these observations, we propose a detection algorithm that makes use of just flow information. Our algorithm has been validated using trusted blacklisting services. The results show that we can detect spamming machines with a 92% accuracy for the traces on which we validate our approach, meaning that the algorithm has a low probability to report false positives (host that are not spammers, but they are flagged as such).

Our work is a first step in flow-based spam detection. In the future, we are interested in assessing the *completeness* of our system, in terms of undetected spamming hosts (false negatives). Moreover, we plan to study how our approach behaves in the presence of very peculiar services, for example a server that is only used for mailing lists. It might happen, indeed, that such systems rank high according to our algorithm, suggesting that other metrics can be added to filter them out. We are also interested in extending our approach to different scenarios, for example Botnet detection.

4.3 Traffic characterization

This sections describes ideas and initial work on traffic characteristics analysis of NREN traffic with regard to application/protocols level in the traffic. NetFlow traces from PIONIER network collected in EMANICS are going to be analyzed which includes especially link to GEANT2 and possibly international Internet providers. An interesting part of this activity would be to compare data received from sampled NetFlow sources in GEANT2 link with a very accurate passive monitoring on this link PSNC has.

The current work includes the definition of different analysis and collection of NetFlow and passive data. The NetFlow flows are being gathered from PIONIER main router as 1 to 10000 samples by flowtools software. We would like to analyze these NetFlow traces by comparing the top N flows, packets and octets with regard to ports used by selected applications/protocols. Then we are going to compare this research results with not sampled passive monitoring on the same link and within the same time period. The interesting point of this research will be also the comparison of the measurement accuracy for the sampled and not sampled measurement methods and their influence on the final results.

Aforementioned passive data monitoring in PIONIER is running on its GEANT2 link (10 Gb/s) using a passive monitoring system developed by CESNET in the scope of EU-funded GN2 project. A central interface - ABW application - provides detailed capacity usage and protocols. It can detect commonly used protocols such as passive FTP (File Transfer Protocol), HTTP (Hypertext Transfer Protocol), Skype telephony or file-sharing protocols, such as BitTorrent, Gnutella or DC++. It also monitors the volume of multicast and IPv6 traffic as two interesting subsets of traffic. Data is available with many resolutions where we will use 1 sec intervals. Data is stored in RRD files although GUI presents data in graphs for quick analysis purposes it is possible to access RRD files directly at the server and dump numeric values. We will analyze total traffic volume as well as separate protocols and different aggregations to show high resolution advantages. This will also be compared with sampled NetFlow to check the accuracy. Then we will take a closer look to commonly used protocols and applications and identify their share in the network traffic.Finally diurnal cycles of NetFlow monitoring from GEANT and Internet will be compared to each other as well as diurnal cycles of NetFlow monitoring from will be compared to passive monitoring to observe differences in monitoring technologies.

Currently NetFlow data is collected using EmanicsLab infrastructure for storage and future processing. Passive data is stored locally in PSNC servers.

5 Management of large MANETs (MaMANET)

5.1 Service Search

In informal data sharing environments, misspellings cause problems for data indexing and retrieval. This is even more pronounced in mobile environments, in which devices with limited input devices are used. In a mobile environment, similarity search algorithms for finding misspelled data need to account for limited CPU and bandwidth. Similarity search for service discovery can significantly improve service management in a distributed environment. As services are often described informally in text form, keyword similarity search can find the required services or data items more reliably. The prototype shows P2P fast similarity search (P2PFastSS) running on mobile phones and laptops that is tailored to uncertain data entry and uses available resources efficiently and P2PFastSS is suitable for similarity search in large-scale network infrastructures, such as service description matching in service discovery or searching for similar terms in P2P storage networks.

5.1.1 Activity

This prototype has been implemented and shown with real devices at the CCNC 2009 conference in the demo session [13] and updated and refined since then. Thus, this text is based on the paper submitted to that conference.

5.1.2 Introduction

Few of the known distributed hash table (DHT)-based search methods support similarity search on keywords. Approximate keyword search is essential, as misspellings and spelling variants make the localization of required information a difficult problem. Without spelling correction, approximately 10% of all queries are not found, because of typos or misspellings [14]. Mobile devices usually have a limited keyboard for textual input, which makes misspellings more likely. Machine learning methods, as applied by Google are not always applicable, as they require a large corpus of queries.

In DHTs, the main operations are *put(key, value)* and *get(key)*. Those operations, in most cases, require $O(\log n)$ messages to be sent in a network with n nodes. Similarity search algorithms for structured P2P networks have been proposed by Ahmed et al. [15], introducing a routing algorithm based on Bloom filters and Wong et al. [16], introducing a routing algorithm in a keyword metric space. However, P2PFastSS [17] is the only algorithm that supports fast similarity searches that runs on existing DHTs without modifying the routing algorithm.

In a real world scenario, a download service on a mobile device, which serves files to other, is offered. Content of this service includes text files or media files with meta information. Other users that are on the same network can use the similarity search to find files. Similarity search is necessary because of misspelled content and user queries using the download service.

5.1.3 Mobile P2P Fast Similarity Search and Demonstration Scenario

For P2PFastSS, the same concept of deletion neighborhood is applied and the neighbors from the target are stored in a DHT. Using P2PFastSS for the example *test* and *fest* would result in the storage of *fest, est, fst, fet, and fes* using the put operation of a DHT.

The following example shows the indexing of the keyword *test* pointing to the document with DocID: 0x321. Keys in *get* and *put* operations are usually generated using a hash function $key = h(string)$. Node 0x1 first generates the neighbors of *test* with $k = 1$ (*test, est, tst, tet, tes*). All neighbors are indexed in the DHT. Figure 5.1.3 shows the indexing of *est*. First, node 0x1 looks for peers with an id close to the id of the neighbor where *est* should be placed, *h(est) = 0x123*. Node 0x4 replies in step 1 with the address of node 0x24, which is closer to 0x123. In step 2, node 0x24 replies with the addresses of nodes 0x124 and 0x122, which are closer to id 0x123. In step 3, these nodes will store the keyword and the document reference for *test*. The keyword is stored redundantly to provide robustness even in case of node failure.

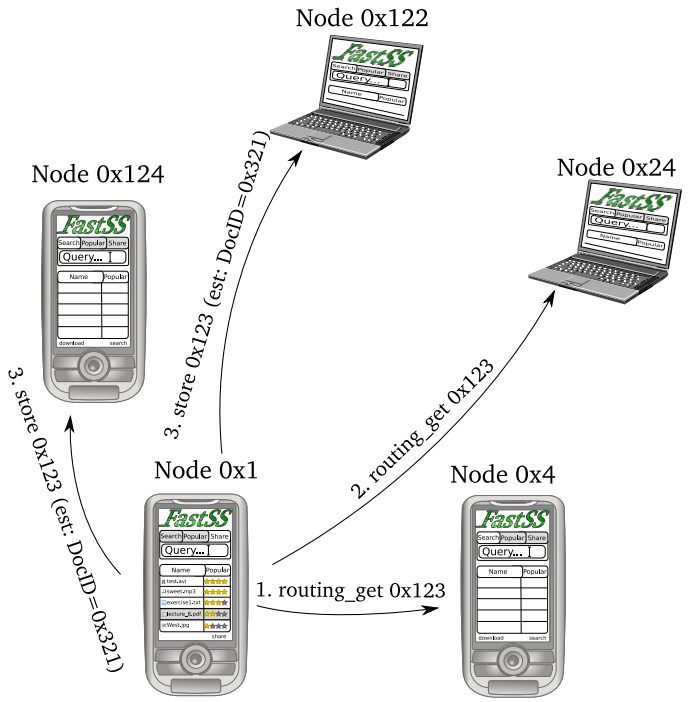


Figure 5: Indexing of *est* with id=0x123

Figure 5.1.3 shows a query execution. Node 0x1 queries for the keyword *fest*. First, neighbors are generated (*fest, est, fst, fet, fes*). In steps 1 and 2, close nodes are queried for neighbor *est*, with id 0x123. In step 3, neighbor 0x124 which stores an index entry for neighbor 0x123 replies with the reference to document 0x321. This document contains the keyword *test*, and as $ed(test, fest) = 1$ the document or a preview of this document is shown to the user.

The demonstrated prototype of mobile P2PFastSs implements the following three layers. The top layer is the user interface for document or content search. The undelying P2PFastSS layer offers two operations, *index* and *search* and carries out neighborhood

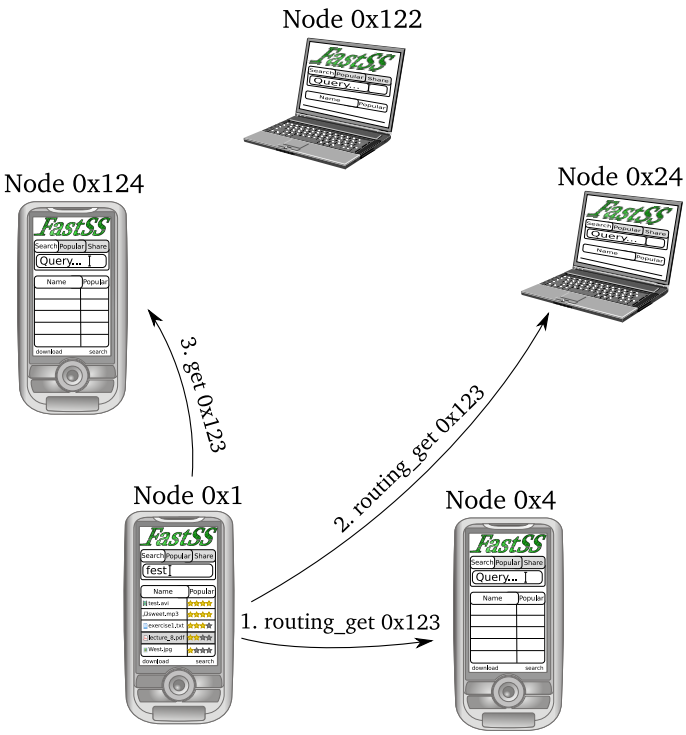


Figure 6: Querying for *est* with id=0x123

generation, indexing and searching. The underlying DHT layer offers *get* and *put* operations. The DHT layer operates on top of Google's Android, which can store data persistently, play media files, and use WiFi.

In this prototype, five users equipped with handsets share documents in a local network. As content, Wikipedia articles and multimedia files are used. Articles are written and indexed by users. From the content, which is published on the local network, keywords are extracted and neighbors are generated. Each neighbor holds references to other devices, subject to a timeout. A user searches for content using keyword search. A keyword search may contain misspellings up to $k = 1$.

A search for a keyword in mobile P2PFastSS returns IPs of mobile devices that host files with a similar keyword. Using P2PFastSS to search for services, the indexed document is the service description. In such a service search scenario, a user that searches for a service retrieves a list of IPs that contain a similar keyword in the service description to the search query. Thus, the presented scenario shows the practicability of mobile P2PFastSS for service discovery and search.

6 Security Management Infrastructure (SeMI)

6.1 Introduction

The secure shell protocol (SSH) [18] is widely used to securely access command line interfaces of network devices and host systems. SSH establishes an encrypted tunnel between a client and a server and allows applications to create several independent channels within the encrypted connection. An SSH server is authenticated by using the server's public/private key pair, also known as the server's hostkey. Clients are authenticated using one of several client authentication methods. The most widely used client authentication methods are the public key, password, and keyboard-interactive authentication methods.

The success of SSH as the protocol of choice to securely access command line interfaces has led to the adoption of SSH as a protocol for other programmatic network management interfaces, namely NETCONF [19, 20] and SNMP [21, 22]. Using the same protocol to securely access interactive management interfaces as well as more programmatic management interfaces reduces the operational costs associated with key management.

The SSH protocol computes new session keys whenever a new SSH session is established. A commonly used SSH key exchange mechanism is based on the Diffie-Hellman algorithm, which is computationally expensive. This introduces significant latency and high processor load when short-lived sessions are established frequently, especially on low-end devices such as DSL routers or wireless access points.

An experimental study of the performance of SNMP over SSH [23, 24] has shown that the overhead associated with SSH session establishment is significant. On a slow machine (Sun Ultra Sparc IIi), an increase of latency by a factor of 15 has been observed for the SSH session establishment and authentication procedure. A significant portion of the delay is caused by the key exchange procedure, i.e., the computation of the session keys. Since many existing SNMP-based management applications and ad hoc scripts do not maintain long lived management sessions, the adoption of SSH as a technology to secure programmatic management interfaces can have significant impact on the CPU load of managed devices and also on the scalability of management applications.

In order to address this problem, we propose a session resumption feature for SSH allowing clients to resume sessions without having to compute new session keys. Session resumption significantly reduces the latency and computational overhead associated with the establishment of SSH session and thus improves the scalability of management applications and reduces the costs of introducing strong security on low-end devices.

The rest of the paper is structured as follows. A short review of SSH is provided in Section 6.2 before we present the proposed session resumption mechanisms in Section 6.3. Section 6.4 discusses some implementation details and Section 6.5 provides an evaluation of the session resumption extension. Related work is discussed in Section 6.6 before we conclude the paper in Section 6.7.

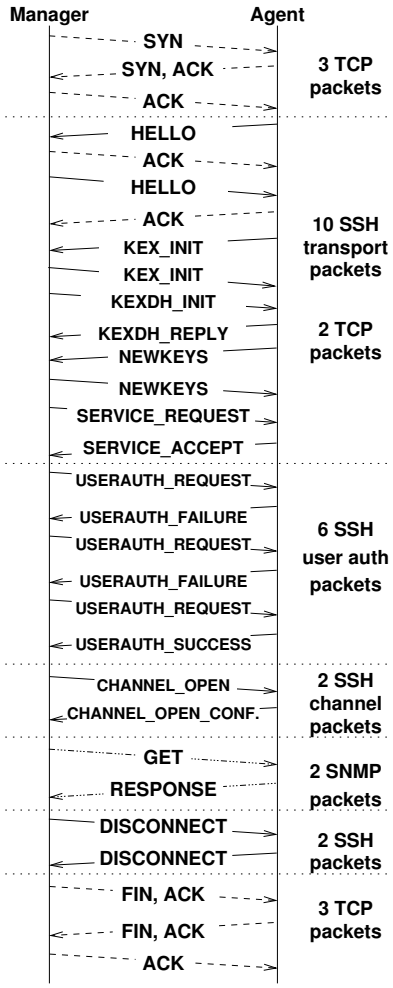


Figure 7: IP packets exchanged by executing an SNMPv3 `GET` operation over SSH

6.2 Secure Shell Protocol

The secure shell protocol (SSH) has a layered architecture [18] consisting of three major components:

- The SSH transport protocol [25] is used to establish a secure communication channel between a server and a client. It provides functions to negotiate message authentication code (MAC), compression, and encryption algorithms, to authenticate the server to the client and to establish session keys.
- The SSH user authentication protocol [26] authenticates the client to the server. It runs over the SSH transport protocol and supports several client authentication methods, such as shared passwords, public keys, or keyboard-interactive (challenge response).
- The SSH connection protocol [27] multiplexes the encrypted connection into several independent logical channels.

A typical SSH exchange carrying a single SNMP `GET` request has been captured on a local area network and is shown in Fig. 7. After the client has established a TCP connection, the

client and the server execute the SSH transport protocol. The first messages exchanged are hello messages identifying among other things the SSH protocol version the client and the server are using.

The next two SSH messages start the key exchange procedure. The server and the client exchange the list of key exchange, server authentication, encryption, MAC, and compression algorithms they support using SSH_MSG_KEXINIT messages. The client and the server select an algorithm and then start the selected key exchange. A key exchange algorithm produces a shared secret K and an exchange hash H. Encryption and authentication keys are derived from K and H.

A widely used key exchange algorithm is the Diffie-Hellman key exchange. The client sends a SSH_MSG_KEXDH_INIT message, which just includes the client's Diffie-Hellman value e. The server returns its value f in a SSH_MSG_KEXDH_REPLY message. In addition, the server signs his reply using his private hostkey. The client then verifies the server's hostkey, usually against a cached copy of the server's public key.

After executing the Diffie-Hellman exchange, the server and the client exchange SSH_MSG_NEWKEYS messages to inform each other that they from now on use the session keys established by the executed key exchange algorithm. From this point on, all communication is encrypted and authenticated using the negotiated algorithms. For security reasons, session keys are renegotiated after a given amount of time has passed or a given amount of data has been transferred. All key negotiations for a session are carried out using the same key exchange algorithm.

In the last step of the SSH transport protocol exchange, the client sends a SSH_MSG_SERVICE_REQUEST message to request a specific service, usually the SSH user authentication service. The client executes the user authentication protocol by iterating through the supported user authentication methods. The first SSH_MSG_USERAUTH_REQUEST usually fails, causing the server to return the list of supported user authentication methods in the SSH_MSG_USERAUTH_FAILURE reply. The client then iterates through the supported methods until a positive reply (SSH_MSG_USERAUTH_SUCCESS) has been received or all methods have been tried.

Using the SSH connection protocol, the client requests to open a channel by sending a SSH_MSG_CHANNEL_OPEN message. The server then responds with a SSH_MSG_ CHANNEL_OPEN_CONFIRMATION message.

Fig. 7 shows the exchange of SNMP GET / RESPONSE messages over the newly created SSH channel before the SSH connection is closed by sending SSH_MSG_DISCONNECT messages, followed by a TCP connection tear-down exchange. Note that TCP ACKs are piggybacked in our example after the initial hello exchange.

6.3 Session Resumption

Session resumption requires that session state — established keys and selected algorithms among other things — is kept for some time after an SSH session has ended.

Clients must certainly maintain their own session state to support session resumption. Server state, on the other hand, could be handled in two different ways. The first option

is to let the server maintain its session state as discussed in Section 6.3.1. The second option is to have the client maintain the server's session state as discussed in Section 6.3.2.

6.3.1 Resumption using Server-Side State

Session resumption using server-side state is straightforward. The server must maintain a session state cache for recently closed SSH sessions. The client and the server indicate their willingness to perform session resumption by negotiating the use of a special session resumption key exchange algorithm. The standard key exchange algorithm negotiation process of SSH can be used for this purpose. If both sides choose to use the session resumption key exchange algorithm, the client sends the session identifier to the server in the first session resumption key exchange message (SSH2_MSG_KEXSR_INIT). The message also contains a MAC computed over the session keys. The server looks up the cached session and verifies the MAC. If successful, it returns an acknowledgement (SSH2_MSG_KEX_SR_OK), followed by a standard SSH2_MSG_NEWKEYS exchange. On failure, a SSH2_MSG_KEX_SR_ERROR is sent and the key exchange proceeds using another key exchange algorithm, or fails. If the session is long-lived, further key negotiations happen using a different key exchange algorithm. Every such negotiation invalidates any previously cached state for that session.

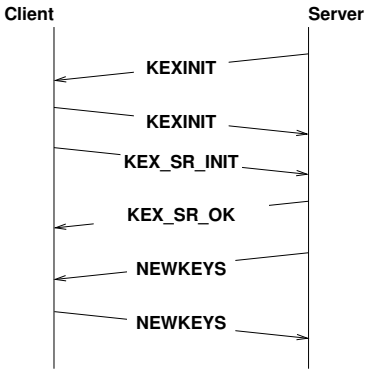
A drawback of this method is that the server has to maintain the session state cache. If a server has to deal with a very large number of clients, then the session state cache can grow out of proportion forcing the server to purge cache entries quickly, thereby reducing the number of successful session resumptions. This problem can be addressed by taking away responsibility for maintaining state from the server entirely, as described below.

6.3.2 Resumption using Client-Side State

Session resumption using client-side state requires introducing a ticket. The ticket is created by the server and contains all information required by the server in order to restore the session state later. After every successful key negotiation that is not itself session resumption, the server generates a ticket, encrypts it, and hands it over to the client in a SSH2_MSG_KEX_SR_TICKET message.

The key used to encrypt a ticket is generated during server start-up, and is known only by the server. Therefore the client must keep state for the session to be resumed beside the ticket, and must be able to correctly associate that state with the corresponding encrypted ticket. If a resumed session is long-lived, further key negotiations happen using an algorithm other than session resumption. Each such renegotiation causes the server to generate and send a new ticket.

During session resumption, the client first sends a SSH2_MSG_KEX_SR_INIT message that contains the encrypted ticket and a MAC over the session identifier. (The identifier is known to the server and to the client, but not to third parties. It is never transmitted unencrypted, in contrast to the server-side state scenario.) The MAC is necessary to ensure that the client is indeed able to restore the session using that ticket, and to prevent



(a) key exchange

Figure 8: Session resumption key exchange

an attacker from intercepting a ticket and using it as its own. If the server decrypts the ticket and verifies the MAC successfully, it sends a `SSH2_MSG_KEX_SR_OK` message, and both sides exchange `SSH2_MSG_NEWKEYS` messages as per the SSH protocol. On failure, a `SSH2_MSG_KEX_SR_ERROR` message indicates that both sides must fall back to another key exchange algorithm, or disconnect.

The structure of the ticket has been designed to only contain the information absolutely necessary to resume a session. Everything else, i.e., all data necessary for a session but not contained in the ticket, is derived during the key exchange procedure in the same way as it is derived for other key exchange algorithms.

A ticket consists of a `TicketContent` structure that is only visible to the server. Its encrypted representation is wrapped in `Ticket` structure, which contains information visible to everyone.

The `TicketContent` structure is shown on Fig. 9. The `session_id` field contains an identifier generated for each session, and `session_id_len` is its length. A resumed session has the same identifier as the one it was resumed from. For each communication direction (client to server and server to client), a session has one algorithm for encryption, one for authentication, and one for compression. These are kept in the next six fields. An encryption algorithm (`TicketEnc`) structure keeps the name of an encryption algorithm, its key, and its initialization vector (IV). The lengths of the key and IV, as well as of the authentication key below, are determined by the algorithm and do not need to be stored. An algorithm is only described by its name under the assumption that both the server and the client will map the name to the same algorithm - a reasonable expectation given that the two used that algorithm for a session not long ago. Similarly, a `TicketMac` structure keeps the name and key of a MAC algorithm. A compression algorithm requires no key, and is only indicated by its name. The `hostkey_type` field indicates the algorithm used to authenticate the server to the client. The server and client version strings provide a reasonable indication that the SSH server or resp. client software has not changed since a ticket was issued. All strings in this structure after `session_id` are NULL-terminated and consequently do not need to have their lengths stored.

A `TicketContent` structure contains very sensitive information and must be protected from third parties, but also from the client. Thus it must be encrypted with a key that is only known to the server. This key and its IV are generated at server start-up. The server is

```
struct TicketEnc {
    char* name;
    u_char* key;
    u_char *iv;
};

struct TicketMac {
    char* name;
    u_char* key;
};

struct TicketContent {
    u_char* session_id;
    u_int session_id_len;
    TicketEnc tenc_ctos;
    TicketEnc tenc_stoc;
    TicketMac tmac_ctos;
    TicketMac tmac_stoc;
    char* tcomp_ctos;
    char* tcomp_stoc;
    int hostkey_type;
    char* client_version_string;
    char* server_version_string;
};
```

Figure 9: `TicketContent` structure

free to choose any encryption algorithm, and any MAC algorithm for ticket ID validation (described below).

```
struct Ticket {
    u_int seq_nr;
    u_char* id;
    u_char* enc_ticket;
    u_int enc_ticket_len;
    int64_t time_stamp;
};
```

Figure 10: `Ticket` structure

What the server actually sends to the client is a `Ticket` structure, shown on Fig. 10. The `seq_nr` field is used by the server if it chooses to keep track of what tickets have been issued. It can then quickly discard some invalid tickets, without attempting decryption or ID validation. Note that `seq_nr` is not related to the SSH sequence numbers used for all packets. The `id` field serves as a unique identifier of that ticket. It is a MAC computed over the key used to encrypt the ticket, the sequence number, and the ticket time stamp. It ensures that the latter two have not been tampered with. The length of `id` is determined by the MAC algorithm in use. The `enc_ticket` field contains the encrypted `TicketContent`, and `enc_ticket_len` is its length. Finally, `time_stamp` indicates the time when the ticket was issued; tickets older than a given threshold can be discarded immediately. It is up to the server to decide how many times and for how long a ticket can be used.

6.3.3 Compatibility Considerations

Care must be taken that the SSH extension interoperates with clients and servers that do not support session resumption. This is achieved by using the key exchange negotiation mechanism of SSH. To support session resumption, we introduce several new key exchange messages. According to [28], the message numbers 30 to 49 are specific to the key exchange method in use and can thus be reused for session resumption key exchange messages [18]. However, in order to obtain session keys, additional messages must be introduced. This requires to go through an IETF consensus process in order to allocated well known message numbers. In the meantime, there is a private number space that can be used for experimentation.

A server supporting client-side state session resumption might send a ticket to a client that does not support tickets. According to the provisions of [25], this should lead to a `SSH_MSG_UNIMPLEMENTED` message and not to a failure.

6.3.4 Security Considerations

Server and client implementations must take care to properly protect their session state caches and/or tickets. This is especially important to clients that must store tickets persistently between executions.

In the server-side state scenario, the session keys are never transmitted, because both sides are assumed to know them. If a third party captures the transmitted session identifier and attempts session resumption using the captures session identifier, the third party will not have the session keys required to produce the correct MAC to validate session resumption.

For client-side state session resumption, session keys are encrypted with a key only known to the server. The client is assumed to know the content of the encrypted ticket already. Non-encrypted portions of the ticket such as the ticket sequence number and timestamp are protected from tampering by the ticket identifier, which is a MAC that requires the server-only key to compute. A third party that captures a ticket would not know the session identifier, because it is encrypted in the ticket. The third party will not be able to produce the correct MAC over the ticket in order to validate the ticket. Even if it could somehow guess the session identifier, it would still not be able to resume the session, because the session keys contained in the ticket are also encrypted.

Because neither side determines any session keys that the other side will use, replay and man in the middle attacks are impossible. Session resumption only affects the SSH transport layer and only restores the encryption and authentication keys used to protect communication from third parties. Both the server and the client must still authenticate each other separately even if a session was just resumed.

6.3.5 Mobility Considerations

The ticket itself does not contain any information related to network identifiers such as IP addresses or port numbers. As such, the ticket itself is not bound to any specific network attachment points and it should be possible to resume SSH sessions even if a client

changes its network attachment point in between. Whether it is feasible to resume SSH sessions where servers change their network attachment point depends on the details of the host authentication and which information is used by the parties involved to identify the server's hostkey.

6.4 Implementation

The session resumption extension described in the previous section has been implemented based on the OpenSSH implementation version 4.7p1. We have implemented session resumption with client-side state. The OpenSSH code-base is separated into three major parts: a static library, a server executable, and a client executable. Session resumption is implemented as a key exchange algorithm in both the server (`srs.c`) and client (`src.c`), with the majority of code shared between them, and therefore placed within the library (`sr.c`). One of our goals has been to minimize the amount of changes necessary to existing OpenSSH source code. Thanks to SSH's layered design, we only had to modify the SSH transport layer implementation, without ever looking at the user authentication or connection layers. Our current implementation consists of about one thousand lines of code of which only 10% are additions or modifications to existing source files.

Normally, the SSH algorithm negotiation procedure described in [25] is used to select an algorithm for key exchange. It is assumed that the algorithm itself cannot fail as long as the underlying transport does not fail, and that it will be used again for long-lived sessions without further negotiation. The most significant change to OpenSSH we had to make is the possibility of graceful failure of a key exchange algorithm, e.g., if an invalid ticket has been used. Upon such failure, both the server and client restart the key exchange with whatever algorithm would have been selected if session-resumption were not available. This fallback procedure does not require the exchange of any additional network packets. It is also used to reset the key exchange algorithm after successful resumption so that long-lived sessions can periodically renegotiate new session keys.

OpenSSH uses a privilege separation model, described in [29]. In this model, for every connection the SSH server daemon forks a process called a "monitor", which runs with full privileges because it must be able to authenticate users. The monitor itself forks unprivileged child processes, which handle all communication with the client. The monitor contains all sensitive information a server must know, and carries out operations that require special privileges on behalf of the child. At any time, there is a set of requests a child is allowed to make; any forbidden request immediately terminates the session. If, through some security exploit, an attacker compromises the server he is communicating with (the unprivileged child), he will not be able to gain privileges on the server machine, because the child that was compromised does not have them.

We made minor modifications to the set of allowed requests, in response to what we believe are minor mistakes in the OpenSSH implementation. These issues had never been triggered before, because all key exchange algorithms other than session-resumption work in essentially the same way. These, along with other minor OpenSSH issues we found, have been reported to the developers.

We note that our current implementation of session resumption does not conform to the privilege separation model, because we give the unprivileged children the secret key used

to sign tickets, instead of giving it to monitors which must then use it on behalf of the children. We did this only as a shortcut to save time and we fully intend to fix this problem; there is no fundamental design limitation that causes the issue.

Serialization for both ticket transfer and storage is done using OpenSSH's `buffer` API. It differs from a straightforward copy of the memory structure only by prepending the length of a string before each string.

The client currently saves a received ticket and the corresponding session keys in two temporary files. For the ticket this is not a problem, but storing the session keys in this way is very insecure. Anyone who obtains them can use them to decrypt a resumed session. Ticket management is a client's responsibility and is not part of the SSH protocol or of our extension. Our basic ticket storage is sufficient as a proof of concept, but it must not be used in a production environment.

We have identified the following important improvements that our current implementation requires:

- Foremost is proper conformance to the privilege separation model. Unprivileged children must not have access to the secret key used to sign tickets.
- Transferring tickets from a server to a client must be made optional and configurable. For example, the server should not try sending tickets if the version string of the client indicates lack of session resumption support, or the client does not advertise session resumption in its set of key exchange algorithms.
- Reasonably configurable secure ticket management for session resumption clients would allow using session resumption in a production environment.
- We must update our implementation against the latest OpenSSH version, which is 5.1 at the time of writing.

Once all of these issues are addressed, we will propose session resumption for inclusion into mainline OpenSSH.

6.5 Evaluation

We have evaluated the session resumption mechanism with client side state by running the command `ssh $host exit` and measuring the overall execution time from the beginning of the SSH exchange until the SSH session has been closed down. The experiments were performed on three Debian GNU/Linux machines (see Table 3). The machines were connected via a switched Gigabit Ethernet with sufficient capacity. The machine called `veggie` was running the SSH client while `meat` and `turtle` acted as SSH servers.

Starting an interactive session using OpenSSH is a relatively expensive operation involving potentially complex things such as creating several processes, password validation, setting up of a shell and a user's environment. The absolute numbers are thus not comparable to other uses of SSH as a secure transport where the overhead of establishing the execution environment is much smaller (e.g., SNMP over SSH).

Table 3: Machines used during the measurements

Name	CPUs	RAM	Ethernet	Kernel
meat	2 Xeon 3 GHz	2 GB	1 Gbps	2.6.16.29
veggie	2 Xeon 3 GHz	1 GB	1 Gbps	2.6.16.29
turtle	1 Ultra Sparc IIi	128 MB	100 Mbps	2.6.20

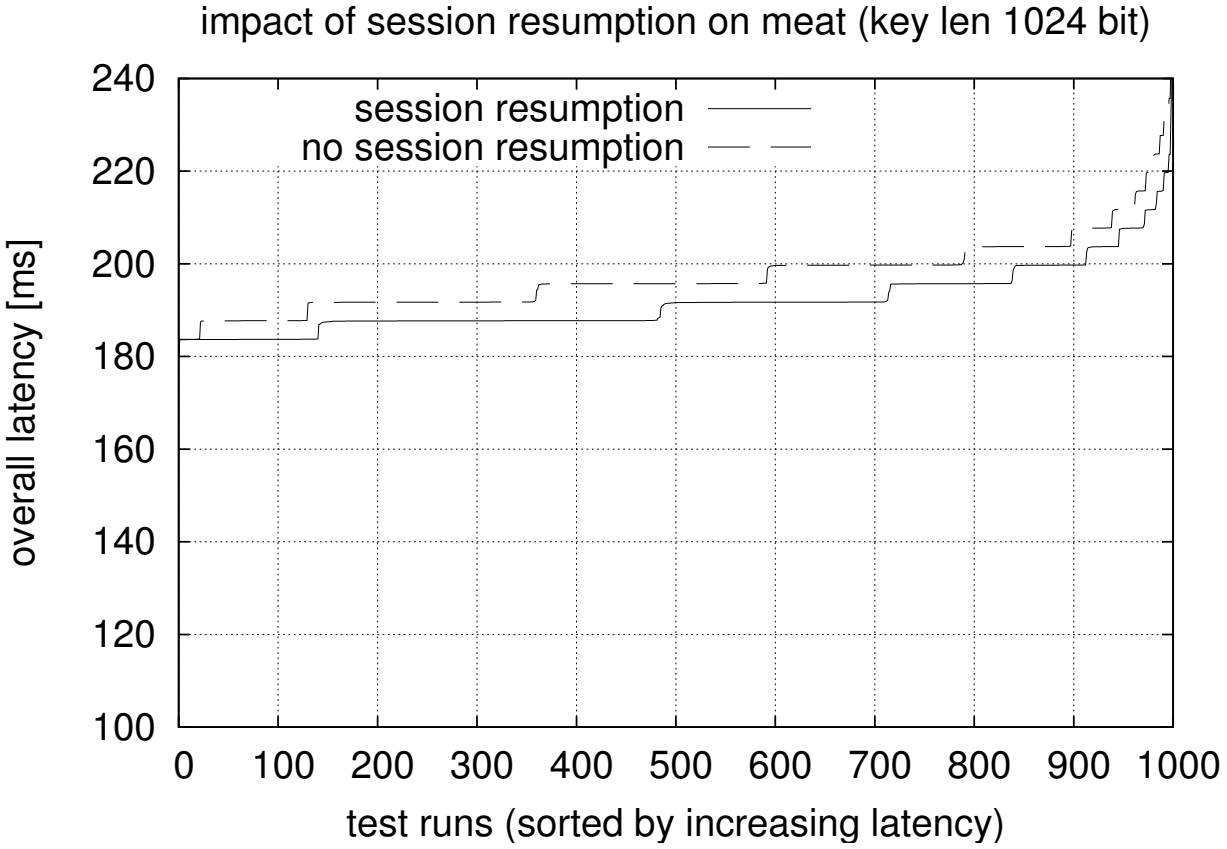


Figure 11: Latency of 1000 ssh sessions with and without session resumption on a fast machine (meat)

During the measurements, we used the hash function HMAC-MD5, the AES-128 encryption algorithm, and the RSA algorithm with a key size of 1024 bits as the public key cryptosystem.

Fig. 11 shows the latency we measured on the fast server meat. We sorted the 1000 data points by the measured latency. The area between the two curves indicates the overall time saving achieved in the 1000 iterations. On meat, $5.4ms$ have been saved on average. Fig. 12 shows the latency we measured on the slow server turtle. On turtle, $310.4ms$ have been saved on average. Obviously, the impact of the slower CPU on the latency is significant and the effect of session resumption becomes much more visible.

It should be noted that we used a key size of 1024 bits, which is still considered secure but likely breakable in a few years. With larger key sizes (2048 bits or 3072 bits), the gains of session resumption will also be visible on faster machines, as shown in Fig. 13. Note that the performance of session resumption is not impacted by the RSA key length.

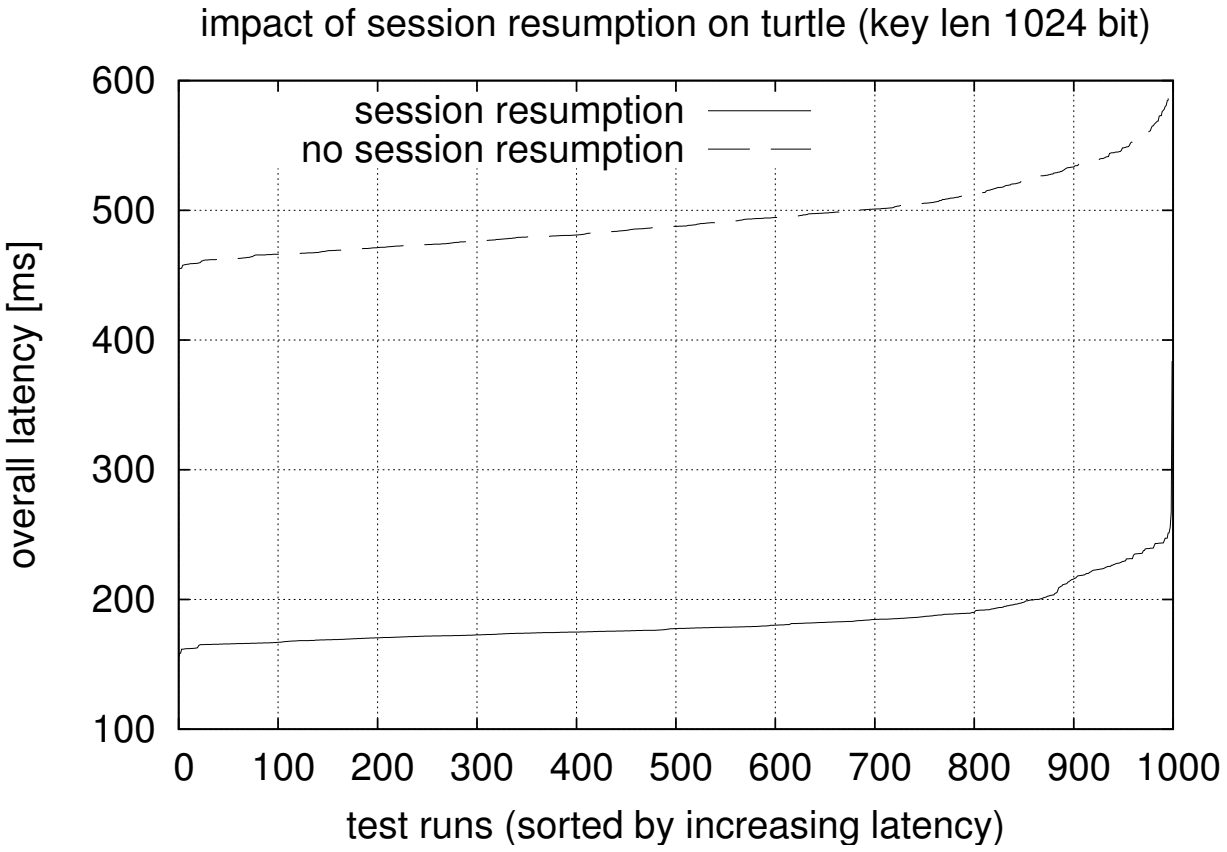


Figure 12: Latency of 1000 ssh sessions with and without session resumption on a slow machine (`turtle`)

6.6 Related Work

Session resumption is a well known concept of the Transport Layer Security (TLS) protocol [30]. The core TLS specification supports session resumption with server-side state. Shachan et al. [31] introduced TLS session resumption with client-side state, which has meanwhile been standardized in [32]. Using self-signed tickets is a well known technique for re-authentication in authentication protocols [33].

The positive effects of TLS session resumption have been studied in [34, 35, 36]. These studies report that the key exchange has a major impact on the performance of TLS and they confirm that session resumption is an effective tool to reduce overhead.

Teemu Koponen et al. [37] extended the SSH and TLS protocols to support resilient connections that can span several sequential TCP connections. They introduce a new key exchange method for SSH called 'resilient', which is similar to our session resumption mechanism with server-side state described in Section 6.3.1. They do not provide a mechanism to resume sessions with client-side state. Another difference is that we do not aim at restoring a session by resynchronizing buffers or compression/encryption state. Instead, our focus is just to restore the session keys of a previously used session.

The OpenSSH implementation supports a feature called connection sharing. With connection sharing enabled, an SSH client connected to a certainer server can act as a

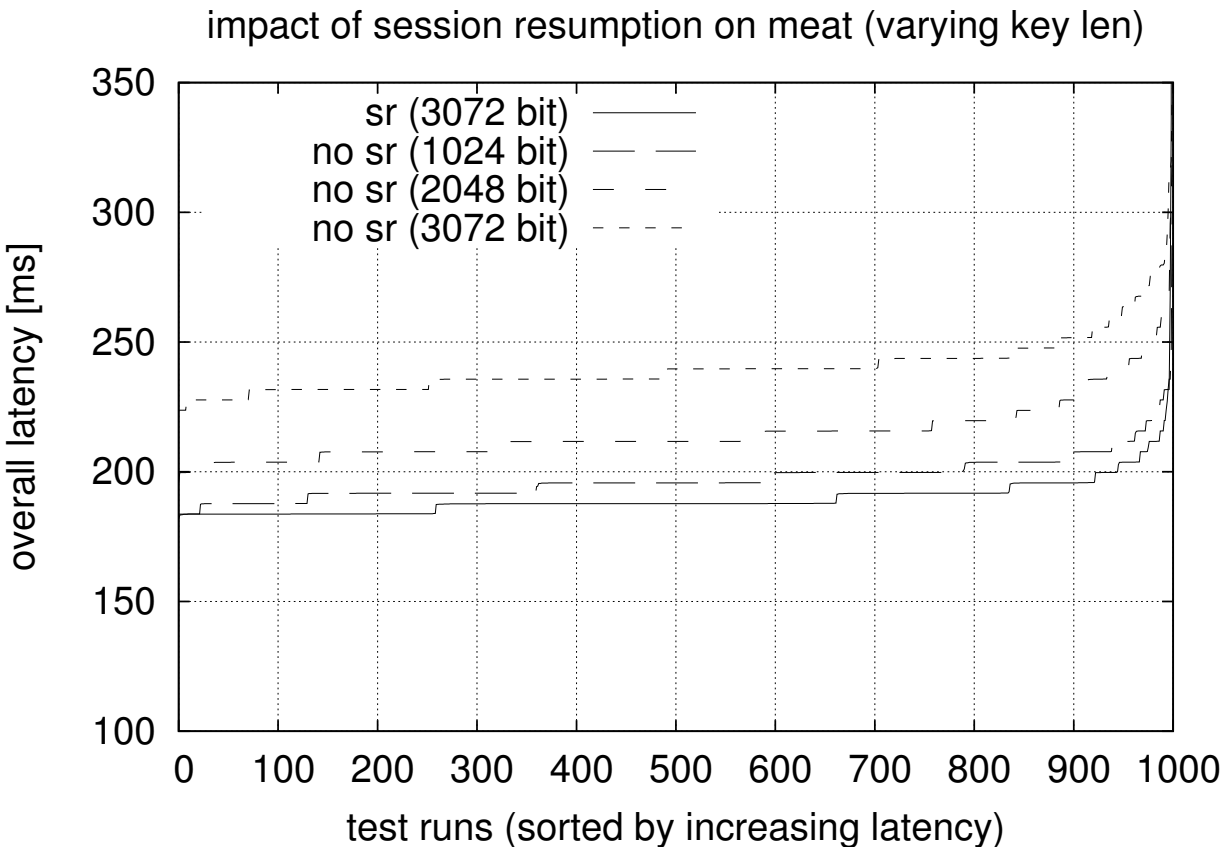


Figure 13: Latency of 1000 ssh sessions with and without session resumption on a fast machine (`meat`), varying the key length

master. Other SSH clients (called slaves) running on the master's host connecting to the same server can ask the master to tunnel the new connection through the master's SSH connection. From the slave's point of view, this is very fast since no key exchange or authentication exchange is needed. Of course, this only works if the slave and master share a trust relationship and a long lasting SSH connection can be maintained.

6.7 Conclusions

The increasing usage of SSH for non-interactive and short-lived sessions requires to take a critical look at its performance. Because early HTTP versions required closing connections to indicate the end of documents, TLS has been optimized for short-lived sessions from the very beginning.

Unlike TLS, SSH has been designed with clear layer separation in mind, at the expense of message complexity and a larger number of round-trips. One significant factor impacting the SSH session establishment performance is the initial session key exchange. The latency introduced by the key exchange algorithm (typically Diffie-Hellman) can be significant, especially on the slow processors commonly used by consumer devices such as wireless access points or DSL routers.

We address this problem by proposing an SSH extension allowing applications to resume previous SSH sessions. The server's session state can be kept either on the server itself, or on the client in the form of a ticket. The solution is backwards compatible and can therefore be deployed gradually.

7 Virtualization Monitoring (VirtMon)

7.1 Introduction

VirtMon Project is structured around four activities, namely Identification of scenarios, Evaluation network setup, Monitoring system approach and Extension to EmancsLab. This section of D7.5 reports on the achievements of each of these activities since the start of the project up to the end of June 2009.

7.2 Activity 1: Identification of scenarios

The objective is to determine virtualization scenarios with quantifiable value addition for commercial/academic situations. In addition, this activity will give us insight about the monitoring parameters that the monitoring system has to track. Up to now only very basic scenarios have been specified. The intention here is not so much to show the capabilities of network virtualization but to ensure that the virtualization mechanism work satisfactorily and to have an initial estimation of the overheads.

7.2.1 Scenario 1

Our first scenario is based in the idea that we can use virtual machines to work as network server at home or at office. In any case, we consider a server that must control the whole network and must have these capabilities:

- Routing
- Fileserver
- Monitoring

Our aim is to compare the efficiency, in terms of CPU, RAM and network traffic, of the solution working with virtualization in respect to the ”non-virtualization” case. Figure 14 shows the testbed realizing the scenario were the Router, the Fileserver and the Monitoring System are installed in different virtual machines (r1t3, Monitor and Sambaserver), in the same physical machine.

In this scenario, we will monitor de CPU load, the RAM and the network interfaces utilization. Network traffic will be exercised in order to study the packet loss and the limitations caused by the CPU load and network interfaces.

7.2.2 Scenario 2

We are working also in a second scenario were the idea is to create some virtual routers on different physical machines. These virtual routers will be linked to create different VPNs. This type of network can be used to offer special services to clients of multimedia

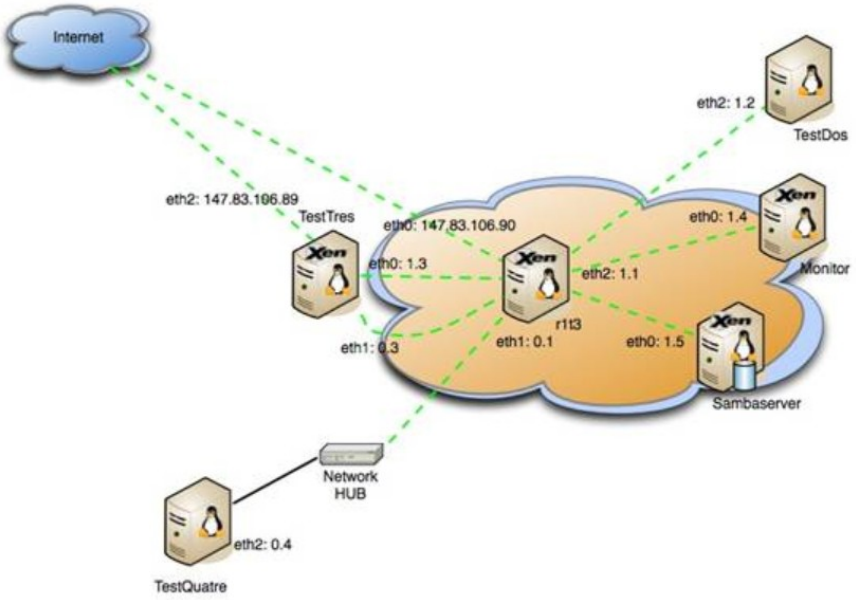


Figure 14: Testbed structure

services like video streaming, music, file sharing or any Internet real time service that needs reserved bandwidth between to machines. The objective is to study how we can use the bandwidth limitations in a Xen environment to give this real time services to different clients.

7.3 Activity 2: Evaluation network setup

This activity is intended to setup a network of virtual routers (Linux routers) and analyze its performance for a given scenario derived from activity 1. More specifically this activity includes the installation and evaluation of routing software intended to run in Linux boxes. Such software can be installed inside a Xen virtual machine, and then the OS and Xen can be configured to ensure all data is passed into the virtual machine that has the software virtual router. Throughput data and flexibility of the virtualized software router will be examined.

At the moment of editing this report, the scenario 1 above mentioned is already installed in a testbed. The Server software such as sambaserver, monitoring system (cacti with SNMP in our case) and routing capabilities are running in each virtual system.

We have verified that all virtual machines have Internet connectivity that pass trough the virtual router (r1t3), and also tested that the packets between TestDos (192.168.1.2) and TestQuatre (192.168.0.4) also pass through the router (r1t3).

Now we must check if traffic between virtual machines in the same sub-network (192.168.1.0/24) also goes to the router and then it routes to each virtual machine or if the bridge created by Xen in native OS (Testres) automatically routes this traffic to the virtual interface that is connected in the bridge.

7.4 Activity 3: Monitoring system approach

This activity is aimed at defining an implementation architecture, which could include at least two alternative approaches of monitoring virtual devices, namely a SNMP-based one and another based on other protocols and tools. More specifically, we plan to envisage, install and configure a monitoring system for virtual machines. In addition we plan to evaluate performance and reliability and perform comparative analysis

In the scenario 1 mentioned above, a monitoring system is already installed and running. The system is based on snmp traps to monitor monitors the CPU Load and traffic in the network interfaces of all machines (virtual and physical) in the Xen environment. As far as the traffic the traffic is concerned we haven't found any problem. With the CPU load, we must check how the percent utilization of CPU in each virtual machine is calculated.

7.4.1 Virtual Machine Monitoring

For the virtual machine monitoring we have defined an architecture that is designed around the concept of producers and consumers. That is, there are producers of monitoring data, which collect data from probes in the system, and there are consumers of monitoring data, which read the monitoring data. The producers and the consumers are connected via a network which can distribute the measurements collected. The architecture itself is based on defined Probes and Data Sources which act as sources of measurement data, and Data Consumers which collect measurements.

Overview In many systems probes are only used to collect data for system management. However, to increase the power and flexibility of the monitoring, given the dynamic nature of virtual machines, we introduce the concept of a Data Source. A data source represents an interaction and control point within the system that encapsulates one or more probes. A probe can send a well defined set of attributes and values to a consumer at a pre-defined interval. The goal for the monitoring system is to have fully dynamic data sources, in which each can have multiple probes, with each probe returning its own data. The data sources will be able to turn on and turn off probes, or change their sending rate. Ideally the data source implementation will have the capability for reprogramming the probes on-the-fly. In this way, it will be possible to make probes send new data if it is required. For example, they can send extra attributes as part of the measurement. A further useful facility for a data source will be the ability to add new probes to a data source at run-time. By using this approach we will be able to instrument components of the system without having to restart them in order to get new information. Such an approach for data sources and probes is important because the monitoring requirements of virtual machines need to grow and adapt for new management requirements over time. If the monitoring system is a fixed point then the management will be limited.

The figure 15 shows a data source with some probes. The probes can be written to collect any kind of data. Also, rather than re-implement everything from scratch, it is clearly beneficial to interface with existing monitoring frameworks in order to collect data. To fit in with the concept of data source and probe, they can be encapsulated with the relevant adapter in the implementation.

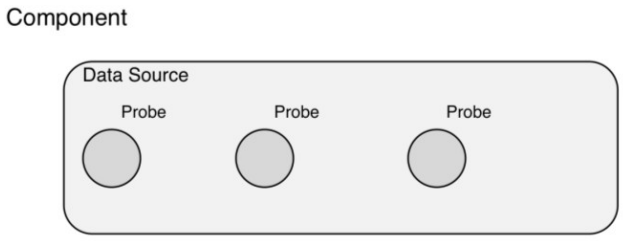


Figure 15: Data source with probes

Implementation We have developed a first working implementation. This implementation acts as a foundation for doing distributed monitoring and provides the framework which binds all of the producers and consumers together.

We have written probes which provide information on the many physical hosts that are used as a platform for running a virtual machine hypervisor. The hypervisor is the core component of many cloud computing environments and is used to run multiple virtual execution environments (VEEs). As well as writing probes to gather data from physical machines, we have evaluated how the distributed monitoring framework can interact with the hypervisor in order to extend the monitoring capabilities to include virtual machines running within the hypervisor. Our work in this area has tried to address the flexible and adaptable requirements of cloud computing and how these fit in with libvirt (the library that provides access to various hypervisors through an abstraction layer). We have made progress in some areas, but have been undermined by the unstable nature of some parts of the libvirt implementation. We are currently investigating how to overcome these issues in order to create a stable hypervisor monitor.

The current design and implementation for monitoring virtual machines is dependent on the fact that a virtual machine can be created, can execute, and can shutdown at run-time whilst the monitoring is still running. In essence, the hypervisor presents a collection of machines that changes over time. From the consumers' point of view, it is important that the virtualized machine should look like an individual host, so we ensure that there is one probe sending data for each virtual machine. The alternative is to have one probe for the whole hypervisor, but using this approach then every measurement will contain data for every virtual machine currently executing on the hypervisor. The consumers will not see individual hosts, but a collection of them. Consequently, virtualized hosts will have to be treated differently from real hosts, which is not a desirable situation.

To ensure that there is one probe per virtual host, and to accommodate the dynamic nature of the hypervisor we have architected a solution shown in figure 16. In the top part of the diagram we can see the hypervisor and the virtualized execution environments (VEEs). To get data from the hypervisor there is a HypervisorController which gets a list of running VEEs from the hypervisor, on a regular basis. The Hypervisor Controller compares the current list of VEEs with the list retrieved last time. From this it determines (a) if there is a new VEE, in which case it asks the HypervisorDataSource to add a new HypervisorProbe for that VEE, or (b) if a VEE has shutdown, in which case it asks the HypervisorDataSource to delete the HypervisorProbe for this VEE.

As stated earlier, a data source represents an interaction and control point within the system, so it is the data source that is responsible for adding and deleting the probes and

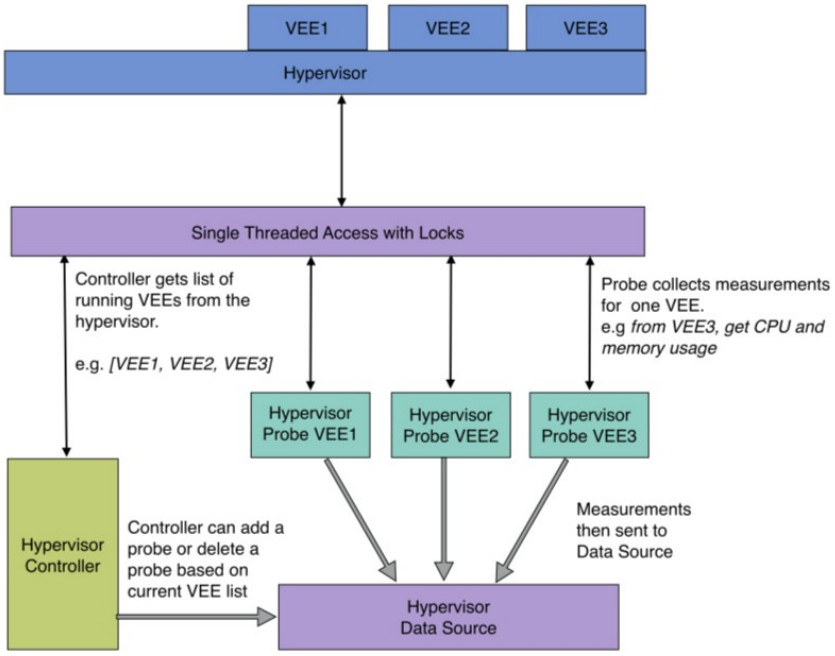


Figure 16: Architecture

to also collect any measurement data from the probes and to pass it onto the network for the consumers. In this case, the HypervisorProbes each run independently of each other, in their own thread, and collect data regarding their assigned VEE at a regular interval. The data collected is structured that same as data collected from a real physical host, such as CPU and memory usage.

The final element of the diagram, is one that provides single threaded access to the hypervisor by using locks. This is strictly necessary as the libvirt implementation is only capable of doing one request at a time, but has no locking or serialization mechanisms of its own. In our multi-threaded implementation, using libvirt directly can cause libvirt to fail.

In conclusion, the hypervisor monitor we have designed and built can collect data from the virtual execution environments in a dynamic and adaptable way. We are currently looking at ways to extend the number of attributes we can collect, such as network throughput for each VEE, and we are looking at the libvirt stability issue.

7.5 Activity 4: Extension to EmanicsLab

This activity is intended to realize in which way the findings of the project could be exported to EmanicsLab for its enhancement. In particular we will investigate the requirements to run the EmanicsLab under Xen. No activity has done yet here.

8 Conclusions

We have reported in this deliverable on the scientific work performed in work-package 7 in the time frame January 2009 to June 2009. This work has been structured around the topics of security, monitoring, and configuration in large scale environments. An open call for activity proposals was initiated at the end of 2008 and finalized at the beginning of 2009. Five activities that cover different aspects of the above mentioned topics have been chosen to be funded during the last phase of the EMANICS project.

In this deliverable, we have described the goals of the five activities as well as the results achieved by them in the reporting period. The reported contents shows that the activities are progressing well. Several subtasks defined by the activities have been completed and have resulted in publications at scientific conferences.

9 Acknowledgment

This deliverable was made possible due to the large and open help of the WP7 Partners of the EMANICS NoE. Many thanks to all of them.

References

[1] A. Sperotto, G. Vliek, R. Sadre, and A. Pras. Detecting spam at the network level. 2009.

[2] Symantec Enterprise Security. The state of spam, a monthly report - February 2009.

[3] Spamassassin. http://spamassassin.apache.org, March 2009.

[4] J. Quittek, T. Zseby, B. Claise, and S. Zander. Requirements for IP Flow Information Export (IPFIX). RFC 3917, NEC Europe, Fraunhofer FOKUS, Cisco Systems, Swinburne University, October 2004.

[5] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A Pras, and B. Stiller. An Overview of IP Flow-based Intrusion Detection. *To appear: IEEE Surverys & Tutorials*, 2009.

[6] A. Ramachandran, N. Feamster, and S. Vempala. Filtering spam with behavioral blacklisting. In *Proc. of the 14th ACM conference on Computer and Communications Security (CCS '07)*, 2007.

[7] D. Schatzmann, M. Burkhart, and T. Spyropoulos. Flow-level Characteristics of Spam and Ham. Technical Report TIK Report Nr. 291, Computer Engineering and Networks Laboratory, ETH, Zurich, August 2008.

[8] D. Schatzmann, M. Burkhart, and T. Spyropoulos. Inferring Spammers in the Network Core. In *Proc. of 10th International Conference on Passive and Active Network Measurement (PAM '09)*, 2009.

[9] P. Desikan and J. Srivastava. Analyzing Network Traffic to Detect E–Mail Spamming Machines. In *Proc. of the 2004 ICDM Workshop on Privacy and Security Aspects of Data Mining (PSDM '04)*, 2004.

[10] B.-C. Cheng, M.-J. Chen, Y.-S. Chu, A. Chen, S. Yap, and K.-P. Fan. SIPS: A Stateful and Flow-Based Intrusion Prevention System for Email Applications. In *Proc. of IFIP International Conference on Network and Parallel Computing (NPC '07)*, 2007.

[11] M. Žádník and Z. Michlovský. Is spam visible in flow-level statistic? Technical report, CESNET technical report 6/2008, 2008.

[12] A. Iverson. Blacklist statistic center. http://stats.dnsbl.com/, March 2009.

[13] Thomas Bocek, Fabio Victora Hecht, Ela Hunt, David Hausheer, and Burkhard Stiller. Mobile P2P Fast Similarity Search. In *6th Annual IEEE Consumer Communications & Networking Conference (CCNC)*, Las Vegas, Nevada, USA, January 2009.

[14] Hercules Dalianis. Evaluating a spelling support in a search engine. In *NLDB*, pages 183–190, 2002.

[15] Reaz Ahmed and Raouf Boutaba. Distributed pattern matching: A key to flexible and efficient p2p search. *IEEE Journal on Selected Areas in Communications*, 25, Issue 1:73–83, January 2007.

[16] Bernard Wong, Aleksandrs Slivkins, and Emin Gn Sirer. Approximate Matching for Peer-to-Peer Overlays with Cubit. Technical Report http://hdl.handle.net/1813/10826, Cornell University, May 2008.

[17] Thomas Bocek, Ela Hunt, David Hausheer, and Burkhard Stiller. Fast Similarity Search in Peer-to-Peer Networks. In *11th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, Salvador, Brazil, April 2008.

[18] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Protocol Architecture. RFC 4251, SSH Communications Security Corp, Cisco Systems, January 2006.

[19] R. Enns. NETCONF Configuration Protocol. RFC 4741, Juniper Networks, December 2006.

[20] M. Wasserman and T. Goddard. Using the NETCONF Configuration Protocol over Secure SHell (SSH). RFC 4742, ThingMagic, ICEsoft Technologies, December 2006.

[21] D. Harrington and J. Schönwälder. Transport Subsystem for the Simple Network Management Protocol (SNMP). Internet Draft (work in progress) <draft-ietf-isms-tmsm-15.txt>, Huawei Technologies (USA), Jacobs University Bremen, November 2008.

[22] D. Harrington, J. Salowey, and W. Hardaker. Secure Shell Transport Model for SNMP. Internet Draft (work in progress) <draft-ietf-isms-secshell-13.txt>, Huawei Technologies, Cisco Systems, Sparta, November 2008.

[23] V. Marinov and J. Schönwälder. Performance Analysis of SNMP over SSH. In *Proc. 17th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2006)*, number 4269 in LNCS, pages 25–36, Dublin, October 2006. Springer.

[24] J. Schönwälder and V. Marinov. On the Impact of Security Protocols on the Performance of SNMP. *(under review)*, 2008.

[25] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Transport Layer Protocol. RFC 4253, SSH Communications Security Corp, Cisco Systems, January 2006.

[26] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Authentication Protocol. RFC 4252, SSH Communications Security Corp, Cisco Systems, January 2006.

[27] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Connection Protocol. RFC 4254, SSH Communications Security Corp, Cisco Systems, January 2006.

[28] S. Lehtinen and C. Lonvick. The Secure Shell (SSH) Protocol Assigned Numbers. RFC 4250, SSH Communications Security Corp, Cisco Systems, January 2006.

[29] N. Provos, M. Friedl, and P. Honeyman. Preventing Privilege Escalation. In *Proc. 12th USENIX Security Symposium*, August 2003.

[30] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346, Independent, RTFM, April 2006.

[31] H. Shacham, D. Boneh, and E. Rescorla. Client-Side Caching for TLS. *ACM Transactions on Information and System Security*, 7(4):553–575, November 2004.

[32] J. Salowey, H. Zhou, P. Eronen, and H. Tschofenig. Transport Layer Security (TLS) Session Resumption without Server-Side State. RFC 5077, Cisco Systems, Nokia, Nokia Siemens Networks, January 2008.

[33] A. Kehne, J. Schönwälder, and H. Langendörfer. A Nonce-Based Protocol for Multiple Authentications. *Operating System Review*, 26(4):84–89, October 1992.

[34] A. Goldberg, R. Buff, and A. Schmitt. Secure Web Server Performance Dramatically Improved by Caching SSL Session Keys. In *Proc. Workshop on Internet Server Performance*, June 1998.

[35] G. Apostolopoulos, V. Peris, P. Pradhan, and D. Saha. Securing Electronic Commerce: Reducing the SSL Overhead. *IEEE Network*, 14(4):8–16, July 2000.

[36] C. Coarfa, P. Druschel, and D. S. Wallach. Performance Analysis of TLS Web Servers. *ACM Transactions on Computer Systems*, 24(1), February 2006.

[37] T. Koponen, P. Eronen, and Mikko Saärelä. Resilient connections for SSH and TLS. In *Proc. of USENIX Annual Technical Conference 2006*, Boston, May 2006.